

On the expressive power of unit resolution

Olivier Bailleux

June 20, 2011

Abstract

This preliminary report addresses the expressive power of unit resolution regarding input data encoded with partial truth assignments of propositional variables. A characterization of the functions that are computable in this way, which we propose to call propagatable functions, is given. By establishing that propagatable functions can also be computed using monotone circuits, we show that there exist polynomial time complexity propagatable functions requiring an exponential amount of clauses to be computed using unit resolution. These results shed new light on studying CNF encodings of NP-complete problems in order to solve them using propositional satisfiability algorithms. A paper is being drafted, which aims to present the concepts introduced in the present report and the underlying scientific issues in a more simple way.

1 Introduction and motivations

Unit resolution is a key feature of state of the art SAT solvers [13] [7] [5], where it speeds up the search for solutions and inconsistencies.

It is well known that different CNF representations of a given problem do not always allow unit resolution to deduce the same information. For example, the CNF encoding for pseudo Boolean constraints proposed in [3] allows unit resolution to restore generalized arc consistency. This is not the case with the encoding proposed in [16], which does not allow unit resolution to deduce as much information as the former encoding does. As a manner of speaking, the expressive power of unit resolution is best exploited using the encoding proposed in [3], with notable consequences on the resolution time.

Two important related questions are "What information can be deduced by unit resolution?" and "Which clauses are required in order to allow this information to be deduced?"

These questions are strongly connected to the characterization of the application field of SAT solvers: "Which problems can be solved as efficiently using a SAT solver as using a specialized solver?" and "How to encode these problems into CNF formulae for optimal resolution time?"

In this paper, we are interested in the functions that can be calculated by means of unit resolution. Studying the expressive power of unit resolution requires characterizing these functions, which will be called *propagatable functions*, and specifying the size of the formulae required to compute them.

Section 2 presents the three main research directions related to the expressive power of unit resolution. Section 3 introduces the concept of propagators and propagatable functions as a formal framework where unit resolution is a computing model. This section also presents theoretical results that will be used in section 4, where the expressive power of unit resolution is compared to the one of monotone Boolean circuits. Section 5 ends the paper with a synthesis of the results, which highlight their implications regarding the efficiency of unit resolution as a filtering technique in SAT solvers.

2 Related works

There are at least three research directions related to the study of the expressive power of unit resolution.

The first one aims to identify the classes of formulae for which unit resolution is a complete refutation procedure in the sense that it produces the empty clause if and only if the input formula is not satisfiable. For example, this property holds for the formulae containing only Horn clauses [9]. This approach differs from that proposed in this paper since it considers the formulae as input data instead of computing systems.

The second direction aims to characterize the complexity of determining whether a given formula can be refuted by unit resolution or not. This decision problem denoted UNIT is known to be P-complete, meaning that for any problem π with polynomial time complexity, there exists a log space reduction from π to UNIT [10]. Circuit value and monotone circuit value, which consist to determine the output value of a Boolean circuit (monotone Boolean circuit, respectively), given its input values, are both P-complete too [8]. Regarding the complexity theory, UNIT, circuit value and monotone circuit value have then the same expressive power. In the present paper, a different point of view is adopted. The CNF formula is not the input data of a program, but the program itself. The input data is a partial truth assignment encoded in a natural way, i.e., each input variable can be either assigned to **true**, assigned to **false**, or not assigned. Similarly, the Boolean circuits are not considered as inputs of a program, but as programs by themselves. In this context, circuit value and monotone circuit value have not the same expressive power for at least two reasons : (1) monotone circuits can only compute monotone Boolean functions, while any Boolean function can be computed using a general Boolean circuit, and (2) there exist monotone Boolean functions which can be computed by polynomially sized Boolean circuits, but requiring an exponential number of gates to be computed using monotone circuits [15]. One of our results is that used as a computation model, unit resolution with natural input encoding has the same expressive power as monotone Boolean circuits, then less expressive power than general Boolean circuits. Obviously, the input encoding plays a central role in this result since by the use of another encoding where all input variables are assigned, unit resolution can easily simulate any Boolean circuit. Nevertheless, the natural encoding is the one used internally in the SAT solvers.

The third line is related to the search for efficient CNF encodings of various problems in order to solve them thanks to a SAT solver. Because unit resolution is implemented efficiently in SAT solvers, many works aim to find encoding schemes which allow unit resolution to make as many deductions as possible. In [6], a CNF encoding for enumerative constraints is proposed, which allows unit propagation to make the same deductions on the resulting formula as restoring arc consistency on the initial constraints does. This work was innovative because with the previously known encodings, unit propagation had less inference power than restoring arc consistency, which is the basic filtering method used in constraint solvers. It has been followed by various similar works on other kinds of constraints such as Boolean cardinality constraints [2] and pseudo-Boolean constraints [4], while in [1], a general way to construct such an encoding for any constraint is proposed. Today, it has become customary, when a new encoding is proposed, to address the question of the behavior of unit resolution on the obtained SAT instances. The problem is that some of these encodings produce a prohibitive number of clauses. This is why some authors seek a trade-off between the size of the formulae and the inference power of unit resolution and other deduction rules implemented in SAT solvers, such as the failed literal rule [11]. For example, this approach is developed in [14] and [12] in the context of Boolean cardinality constraints.

3 Propagatable functions

3.1 Unit resolution

This section recalls the terminology and the principles involved in unit resolution, and introduces the notations that will be used in the rest of the paper.

A *literal* is either a propositional variable or a negated propositional variable. A CNF *formula* is

defined as a conjunction $c_1 \wedge \dots \wedge c_k$ of *clauses*, where each clause $c_i = l_{i,1} \vee \dots \vee l_{i,|c_i|}$ is a disjunction of literals. The *size* of a CNF formula is its number of literal occurrences.

A *truth assignment* on a set of propositional variables is a function mapping some of the variables in this set to truth values, i.e., **true** or **false**. These variables are said to be *fixed* to **true** or **false**. If a truth assignment does not fix all the variables, it is said to be *partial*; else it is said to be *complete*. In this paper, a truth assignment will be represented as a set of literals. Given a propositional formula ϕ and a truth assignment I , $\phi|_I$ denotes $\phi \bigwedge_{l \in I} (l)$.

Any CNF formula ϕ is said to be *satisfied* (falsified, respectively) by a truth assignment I if and only if I causes ϕ to evaluate to **true** (**false**, respectively) in the standard way. A CNF formula ϕ is said to be *satisfiable* if and only if there exists a truth assignment that satisfy ϕ . Any complete truth assignment satisfying a CNF formula ϕ is called a *model* of ϕ .

For convenience, a clause can be represented as a set of literals and a CNF formula can be represented as a set of clauses.

Example 1. The CNF formula $(a \vee \bar{b}) \wedge (\bar{a} \vee b)$ can be represented as $\{\{a, \bar{b}\}, \{\bar{a}, b\}\}$.

Unit resolution is an inference technique which aims either to detect an inconsistency or to assign some variables, so as to simplify a CNF formula. As described in a standard way by Algorithm 1, until there is no empty clause and there is at least one unit clause (w) in the input formula, all the occurrences of the literal \bar{w} and all the clauses containing the literal w are removed.

```

input :  $\phi$  [CNF formula];
output:  $(\phi, E)$  [(CNF formula, set of literals)] or  $\perp$ ;
 $E \leftarrow \{\}$ ;
while  $\{\} \notin \phi$  and  $\exists \{l\} \in \phi$  do
    |  $\phi \leftarrow \phi \setminus \{c : c \in \phi, l \in c\} \setminus \{c : c \in \phi, \bar{l} \in c\} \cup \{c \setminus \{\bar{l}\} : c \in \phi, \bar{l} \in c\}$ ;
    |  $E \leftarrow E \cup \{l\}$ ;
end
if  $\{\} \in \phi$  then
    | return  $\perp$ 
else
    | return  $(\phi, E)$ 
end

```

Algorithm 1: The standard algorithm for unit resolution

In the following, we will consider another algorithm (Algorithm 3) that will be called the *alternative algorithm for unit resolution*. This alternative algorithm return \perp if and only if the standard algorithm return \perp , else it returns the same truth assignment as the standard algorithm does. But contrary to the standard algorithm, it does not modify the input formula.

The alternative algorithm repeat $n + 1$ *propagation stages*, where n is the number of variables in the input formula. Each of these propagation stage is performed by the procedure **propagation** (Algorithm 2).

The standard and the alternative algorithms are strictly equivalent. The first one produces a literal w if there is a unit clause (w) in the simplified formula, that is if there is a clause $(l_1 \vee \dots \vee l_q \vee w)$ in the input formula such that the literals $\bar{l}_1, \dots, \bar{l}_q$ are previously produced. The second one produces the same literal in the same conditions, with the only difference that it does not modify the input formula. The standard algorithm return \perp when an empty clause is produced. This occurs when there is some unit clause (w) and the opposite clause (\bar{w}) in the simplified formula. In the same situation, the alternative algorithm does not stop, but adds both the literals w and \bar{w} in the set E . As the standard algorithm, it will return \perp at the end of its execution.

```

input :  $\phi, E$  [(CNF formula, set of literals)];
output:  $F$  [set of literals];

 $F \leftarrow \{\}$ ;
foreach literal  $w$  in  $\phi$  such that  $w \notin E$  do
  | foreach clause  $(l_1 \vee \dots \vee l_k \vee w)$  of  $\phi$  such that  $\bar{l}_1, \dots, \bar{l}_k \in E$  do
  | |  $F \leftarrow F \cup \{w\}$ 
  | end
end
return  $F$ ;

```

Algorithm 2: The procedure **propagation**, which performs a propagation stage of the alternative algorithm for unit resolution.

```

input :  $\phi$  [CNF formula];
output:  $E$  [set of literals] or  $\perp$ ;

 $E \leftarrow \{\}$ ;
 $n \leftarrow$  the number of variables in  $\phi$ ;
repeat  $n + 1$  times
  |  $E \leftarrow E \cup \text{propagation}(\phi, E)$ ;
end
if there exists  $v$  such that  $v, \bar{v} \in E$  then
  | return  $\perp$ 
else
  | return  $E$ 
end

```

Algorithm 3: The alternative algorithm for unit resolution

Of course, the alternative algorithm could be optimized in such a way that it stops if the last propagation stage did not modify the set E , or if there are a literal w and its opposite \bar{w} in E . One of these two events necessarily occurs during the first $n + 1$ propagation stages, because if E contains $n + 1$ literals, it necessarily contains two opposite literals. This optimization has not been done because this algorithm is not intended to be implemented. It is only a way to prove some theoretical results which will be presented later.

In the following of the paper, given any CNF formula ϕ with n variables and any integer $1 \leq k \leq n + 1$,

- $\mathcal{U}_k(\phi)$ will denote the set of literals produced at the k^{th} propagation stage of algorithm 3;
- $\mathcal{U}_{1..k}(\phi)$ will denote $\cup_{i=1}^k \mathcal{U}_i(\phi)$;
- $\mathcal{U}(\phi)$ will denote the result of unit propagation applied to ϕ , i.e., either \perp or $\mathcal{U}_{1..n+1}(\phi)$.

Example 2. With $\phi = (a \vee \bar{b}) \wedge (b) \wedge (\bar{a} \vee c \vee \bar{d})$ as input, both algorithms 1 and 3 return $\mathcal{U}(\phi) = \{b, a\}$. Regarding algorithm 3, $\mathcal{U}_1(\phi) = \{b\}$ and $\mathcal{U}_2(\phi) = \{a\}$.

3.2 Reified formulae

This section introduces the notion of reified CNF formula, which will be subsequently used as a tool to prove several results.

Informally speaking, the reified counterpart of any CNF formula ϕ , is a *satisfiable* CNF formula $\sigma = \text{reif}(\phi)$ such that applying unit resolution on σ simulates all the inferences produced by applying unit resolution on ϕ .

Let $\text{var}(\phi)$ denote the set of the variables occurring in ϕ .

Definition 1 (reified formula). *Let ϕ be a CNF formula with n variables. The reified counterpart of ϕ is the formula $\sigma = \text{reif}(\phi)$ obtained as follows:*

- *There are $2n(n+2)$ variables in σ , namely*

$$\text{var}(\sigma) = \cup_{v \in \text{var}(\phi)} \{v_0^+, v_0^-, \dots, v_{n+1}^+, v_{n+1}^-\}$$

Given any propositional variable $v \in \text{var}(\sigma)$, let $\delta(v)$ denotes v^+ and $\delta(\bar{v})$ denotes v^- .

- *σ consists of the following clauses:*

- (1) *for any unary clause (w) of ϕ , $(\delta(w)_0) \wedge (\overline{\delta(w)_0} \vee \delta(w)_1)^1$, which will be called initialization clauses of rank 0 and 1,*
- (2) *for any stage $2 \leq i \leq n+1$, and any variable v of ϕ , $(\overline{v_{i-1}^+} \vee v_i^+) \wedge (\overline{v_{i-1}^-} \vee v_i^-)$, which will be called propagation clauses of rank i ,*
- (3) *for any stage $2 \leq i \leq n+1$, and for any non-unary clause q of ϕ , $\bigwedge_{w \in q} \chi(q, i, w)$, where*

$$\chi(q, i, w) = \delta(w)_i \vee \bigvee_{t \in q, t \neq w} \overline{\delta(t)_{i-1}},$$

which will be called deduction clauses of rank i .

For convenience, in the following of the paper, each time that a formula ϕ and its reified counterpart $\sigma = \text{reif}(\phi)$ will be considered, the propagation stages on ϕ will be numbered from 1, while the propagation stages on σ will be numbered from 0.

Example 3. *Let $\phi = (a) \wedge (\bar{a} \vee b)$. The reified counterpart of ϕ can be decomposed as $\sigma = \sigma_{(1)} \wedge \sigma_{(2)} \wedge \sigma_{(3)}$, where:*

$$\sigma_{(1)} = (a_0^+) \wedge (\overline{a_0^+} \vee a_1^+)$$

$$\sigma_{(2)} = (\overline{a_1^+} \vee a_2^+) \wedge (\overline{a_1^-} \vee a_2^-) \wedge (\overline{b_1^+} \vee b_2^+) \wedge (\overline{b_1^-} \vee b_2^-) \wedge (\overline{a_2^+} \vee a_3^+) \wedge (\overline{a_2^-} \vee a_3^-) \wedge (\overline{b_2^+} \vee b_3^+) \wedge (\overline{b_2^-} \vee b_3^-)$$

$$\sigma_{(3)} = (\overline{a_1^+} \vee b_2^+) \wedge (\overline{b_1^-} \vee a_2^-) \wedge (\overline{a_2^+} \vee b_3^+) \wedge (\overline{b_2^-} \vee a_3^-)$$

*The stage 1 of unit resolution fixes a to **true** in ϕ . Accordingly, thanks to the initialization clauses, the stages 0 and 1 of unit resolution fix a_0^+ and a_1^+ to **true** in σ .*

*At the stage 2, unit resolution fixes b to **true** in ϕ . Accordingly, at the stage 2 of unit resolution of σ , the variable b_2^+ is fixed to **true** thanks to the deduction clause $(\overline{a_1^+} \vee b_2^+)$, and the variable a_2^+ is fixed to **true** thanks to the propagation clause $(\overline{a_1^+} \vee a_2^+)$.*

*At the stage 3, unit resolution fixes no new variable in ϕ . Thanks to the propagation clauses $(\overline{a_2^+} \vee a_3^+)$ and $(\overline{b_2^-} \vee b_3^-)$, the stage 3 of unit resolution on σ fixes a_3^+ and b_3^- to **true**.*

This example shows the roles of the three kind of clauses. The initialization clauses simulate the first stage of unit resolution, which consists to fix the variables occurring in unit clauses. The propagation clauses allow unit resolution to propagate the values that were previously assigned. For example, if the variable a is fixed to **true** in ϕ at stage 1, i.e., a_1^+ is fixed to **true** in σ , then the clause $(\overline{a_1^+} \vee a_2^+)$ ensures that a_2^+ is fixed to **true** in σ at stage 2. The deduction clauses allow unit resolution to simulate in σ the deductions that are made in ϕ . For example, if at stage 1, the variable a is fixed to **true** in ϕ ,

¹ $(\delta(w)_1)$ would have the same effect in only one propagation stage, but the reified formula is intentionally tailored in such a way that the variables v_i^+ and v_i^- are fixed at the $(i+1)^{\text{th}}$ propagation stage.

and if there is a clause $(\bar{a} \vee b)$ in ϕ , then the clause $(\overline{a_1^+} \vee b_2^+)$ of σ allows unit resolution to fix b_2^+ to **true**, which indicates that unit resolution fixes b to **true** in ϕ .

Note that the proposed model of reified formula is not optimal in the sense that it usually involves redundant clauses and useless clauses. Our purpose is to provide a tool for proving theoretical results which will be presented in the following of the paper. In this context, the relevant property of the reified counterpart of a formula ϕ is that its size is polynomially related to the size of ϕ . Namely, if there are n variables and k clauses of size at most p in ϕ , then there are $O(n^2k)$ clauses of size at most p in the reified counterpart σ of ϕ , because for any of the $O(n)$ propagation stages, σ contains $O(n)$ propagation clauses and $O(kn)$ deduction clauses², and because the number of literals in any clause of σ cannot exceed the size of the longest clause of ϕ .

Definition 2 (reified formula with injected variables). *Let ϕ be any CNF formula with n variables and $V \subseteq \text{var}(\phi)$ be a set of propositional variables. The formula*

$$\text{reif}(\phi, V) = \text{reif}(\phi) \wedge \left(\bigwedge_{v \in V} ((\bar{v} \vee v_1^+) \wedge (v \vee v_1^-)) \right)$$

is said to be the reified counterpart of ϕ with injected variables V . The clauses added to $\text{reif}(\phi)$ will be called injection clauses.

Lemma 1. *Let ϕ be any CNF formula with n variables. Let $\sigma = \text{reif}(\phi)$ and i be any integer such that $0 \leq i \leq n+1$. For any variable $v \in \text{var}(\phi)$, applying unit resolution on σ can fix v_i^+ and/or v_i^- only at the propagation stage i , and only to **true**.*

Proof. This property can be proved by induction on i . The only variables which can be fixed at the first propagation stage, i.e. $i=0$, are in $\{v_0^+, v_0^-, v \in \text{var}(\phi)\}$, because no other variables are in unary clauses, and these variables can only be fixed to **true**, because they occur positively. Now, given any $1 \leq i \leq n+1$, let us suppose the property hold until the $(i-1)^{\text{th}}$ propagation stage (which implies that no variable in $\{v_i^+, v_i^-, 0 \leq i \leq i-1, v \in \text{var}(\phi)\}$ has been fixed negatively). The only way for unit resolution to fix variables in $\{v_i^+, v_i^-, v \in \text{var}(\phi)\}$ is through clauses involving variables in $\{v_{i-1}^+, v_{i-1}^-, v \in \text{var}(\phi)\}$, which, by induction hypothesis, can only be fixed at the propagation stage $i-1$. Because the variables in $\{v_{i-1}^+, v_{i-1}^-, v \in \text{var}(\phi)\}$ occur positively in these clauses, they can be only fixed to **true**. \square

Theorem 1. *Let ϕ be any CNF formula with n variables and $\sigma = \text{reif}(\phi)$ be the reified counterpart of ϕ . The following properties hold:*

1. σ is satisfiable.
2. For any variable $v \in \text{var}(\phi)$, and any integer $1 \leq k \leq n+1$, the two following properties hold:

- (a) $v_k^+ \in \mathcal{U}_k(\sigma)$ if and only if $v \in \mathcal{U}_{1..k}(\phi)$;
- (b) $v_k^- \in \mathcal{U}_k(\sigma)$ if and only if $\bar{v} \in \mathcal{U}_{1..k}(\phi)$.

Proof. The first property arises because each clause of σ contains at least one positive literal.

The second property can be proved by induction on k . For sake of brevity, let us reformulate it as follows: for any variable $v \in \text{var}(\phi)$, any integer $1 \leq k \leq n+1$, and any literal $w \in \{v, \bar{v}\}$, $\delta_k(w) \in \mathcal{U}_k(\sigma)$ if and only if $w \in \mathcal{U}_{1..k}(\phi)$.

Clearly, the property holds for $k=1$ because there is a clause (w) in ϕ if and only if there is a clause $(\delta_0(w))$ and a clause $(\overline{\delta_0(w)} \vee \delta_1(w))$ in σ . Now let us suppose that the property holds until the propagation stage $k-1$, $k > 1$.

²Because each of the n variables occurs at most in k clauses.

\Rightarrow

Let us suppose that $\delta_k(w) \in \mathcal{U}_k(\sigma)$. Then, according to the lemma 1, one of the two following conditions hold:

1. There is a propagation clause $(\overline{\delta_{k-1}(w)} \vee \delta_k(w))$ in σ and $\delta_{k-1}(w) \in \mathcal{U}_k(\sigma)$. By induction hypothesis, $w \in \mathcal{U}_{1..k-1}(\phi)$, then $w \in \mathcal{U}_{1..k}(\phi)$.
2. There is a deduction clause $(\overline{\delta_{k-1}(\bar{l}_1)} \vee \dots \vee \overline{\delta_{k-1}(\bar{l}_q)} \vee \delta_k(w))$ in σ and $\delta_{k-1}(\bar{l}_1), \dots, \delta_{k-1}(\bar{l}_q) \in \mathcal{U}_{k-1}(\sigma)$. This means that there is a clause $(l_1 \vee \dots \vee l_q \vee w)$ in ϕ , and, by induction hypothesis, $\bar{l}_1, \dots, \bar{l}_q \in \mathcal{U}_{1..k-1}(\phi)$. Then $w \in \mathcal{U}_{1..k}(\phi)$.

\Leftarrow

Let us suppose that $w \in \mathcal{U}_{1..k}(\phi)$. Then, according to the principle of unit resolution, one of the two conditions hold:

1. There is a clause (w) in ϕ . Then, as shown in the first part of this proof, $\delta_1(w) \in \mathcal{U}_1(\sigma)$. Thanks to the propagation clauses $(\overline{\delta_1(w)} \vee \delta_2(w)), \dots, (\overline{\delta_{k-1}(w)} \vee \delta_k(w))$, $\delta_k(w) \in \mathcal{U}_k(\sigma)$.
2. There is a clause $(l_1 \vee \dots \vee l_q \vee w)$ in ϕ and an integer $1 \leq i \leq k-1$ such that $\bar{l}_1, \dots, \bar{l}_q \in \mathcal{U}_{1..i}(\phi)$. By construction of σ , there is a deduction clause $(\overline{\delta_i(\bar{l}_1)} \vee \dots \vee \overline{\delta_i(\bar{l}_q)} \vee \delta_{i+1}(w))$ in σ , and by induction hypothesis, $\delta_i(\bar{l}_1), \dots, \delta_i(\bar{l}_q) \in \mathcal{U}_i(\sigma)$. Then $\delta_{i+1}(w) \in \mathcal{U}_{i+1}(\sigma)$. Either because $i+1 = k$ or thanks to the propagation clauses $(\overline{\delta_i(w)} \vee \delta_{i+1}(w)), \dots, (\overline{\delta_{k-1}(w)} \vee \delta_k(w))$, $\delta_k(w) \in \mathcal{U}_k(\sigma)$.

□

As an interesting corollary of Theorem 1, the failed literal rule [11], which is a speed up technique implemented in some modern SAT solver, can be simulated by unit propagation. Given a formula σ and a literal l , the failed literal rule aims to test if l must be fixed. Unit resolution is applied to $\sigma \wedge (\bar{l})$ ($\sigma \wedge (l)$, respectively). If an inconsistency is detected then l (\bar{l} , respectively) is fixed to **true**. The same result can be obtained by applying unit resolution on $\text{reif}(\phi \wedge (l)) \wedge \bigwedge_{v \in \text{var}(\phi)} (\overline{v_{n+1}^+} \vee \overline{v_{n+1}^-} \vee \bar{l})$ ($\text{reif}(\phi \wedge (\bar{l})) \wedge \bigwedge_{v \in \text{var}(\phi)} (\overline{v_{n+1}^+} \vee \overline{v_{n+1}^-} \vee l)$, respectively).

Theorem 2. *Let ϕ be any CNF formula, $V \subseteq \text{var}(\phi)$ be a set of propositional variables, and $I \in \mathcal{I}_V$ a truth assignment. $\mathcal{U}((\text{reif}(\phi, V))|_I) = \mathcal{U}(\text{reif}(\phi|_I))$, i.e., unit resolution have the same effect on $\mathcal{U}((\text{reif}(\phi, V))|_I)$ as it does on $\mathcal{U}(\text{reif}(\phi|_I))$.*

Proof. For any literal $v \in I$ ($\bar{v} \in I$, respectively), the two first stages of unit resolution applied to $(\text{reif}(\phi, V))|_I$ produces the literals v_1^+ (v_1^- , respectively), which are the literals produced from the clauses $(\overline{v_0^+} \vee v_1^+)$, (v_0^+) ($(\overline{v_0^-} \vee v_1^-)$, (v_0^-) , respectively) of $\text{reif}(\phi|_I)$. The other stages of unit resolution behave similarly in the two formulae, because the same clauses are involved. □

3.3 Computing with unit resolution

This section explains how unit resolution can be used to compute functions.

3.3.1 Definitions and terminology

Definition 3 (Propagator). *Let ϕ be a CNF formula, $\text{var}(\phi)$ be the set of propositional variables occurring in ϕ , $V \subseteq \text{var}(\phi)$ a set of propositional variables, and $s \in V$ a propositional variable. The triplet $P = \langle \phi, V, s \rangle$ is called a propagator. The size of P is the size of the formula ϕ .*

A propagator $\langle \phi, V, s \rangle$ can act as a computer in the following way:

- the input data is a partial truth assignment I of the variables in V , i.e., some variables are assigned to **true**, some are assigned to **false**, and the other are not assigned,
- the output can take four possible values according to the result of applying unit resolution:
 - **fail** if $\mathcal{U}(\phi|_I) = \perp$,
 - **true** if $\mathcal{U}(\phi|_I) \neq \perp$ and $s \in \mathcal{U}(\phi|_I)$,
 - **false** if $\mathcal{U}(\phi|_I) \neq \perp$ and $\bar{s} \in \mathcal{U}(\phi|_I)$,
 - **na** if $\mathcal{U}(\phi|_I) \neq \perp$ and $\bar{s} \notin \mathcal{U}(\phi|_I)$ and $s \notin \mathcal{U}(\phi|_I)$.

Formally, $\langle \phi, V, s \rangle$ computes a function f with domain \mathcal{I}_V and codomain $\{\mathbf{fail}, \mathbf{true}, \mathbf{false}, \mathbf{na}\}$, where \mathcal{I}_V denotes the set of all the consistent partial assignments on V , i.e., $\{I \subset V \cup \{\bar{v}, v \in V\}, \forall I \in \mathcal{I}_V, \bar{I} \notin \mathcal{I}_V\}$.

Conversely, given a set V of propositional variables and any function f with domain $D \subseteq \mathcal{I}_V$ and codomain $\{\mathbf{fail}, \mathbf{true}, \mathbf{false}, \mathbf{na}\}$, the following issues can be addressed:

1. Can f be computed by a propagator ?
2. If yes, how many clauses are required to compute f using unit resolution ?

These questions are important because in a SAT solver, unit resolution is used both for detecting inconsistencies (the **fail** answer) and for inferring new information (the **true** or **false** answers), with the effect of accelerating the resolution. It is then useful to use concise CNF encodings which allow unit resolution to achieve as many deductions as possible.

Definition 4 (reified propagator). *Let $P = \langle \phi, V, s \rangle$ be a propagator. The reified counterpart of P is $\text{reif}(P) = \langle \psi, V, s^{\mathbf{true}}, s^{\mathbf{false}}, s^{\mathbf{fail}} \rangle$, such that $s^{\mathbf{true}}$ is the variable s_{n+1}^+ , $s^{\mathbf{false}}$ is the variable s_{n+1}^- , $s^{\mathbf{fail}}$ is new fresh variable, and*

$$\psi = \text{reif}(\phi, V) \wedge \left(\bigwedge_{u \in \text{var}(\phi)} (\overline{u_{n+1}^+} \wedge \overline{u_{n+1}^-} \wedge s^{\mathbf{fail}}) \right)$$

By construction of the formula ψ , given any $I \in \mathcal{I}_V$, applying unit resolution to $\psi|_I$ never returns \perp and simulates unit resolution on $\phi|_I$ in the following sense:

- unit resolution on $\phi|_I$ returns \perp if and only if unit resolution on $\psi|_I$ produces $s^{\mathbf{fail}}$ (i.e. fixes to $s^{\mathbf{fail}}$ to **true**);
- unit resolution on $\phi|_I$ produces s if and only if unit resolution on $\psi|_I$ produces $s^{\mathbf{true}}$;
- unit resolution on $\phi|_I$ produces \bar{s} if and only if unit resolution on $\psi|_I$ produces $s^{\mathbf{false}}$.

Definition 5 (filtering function). *Let V be a set of propositional variables. Any function f with domain $D \subseteq \mathcal{I}_V$ and codomain $\{\mathbf{fail}, \mathbf{true}, \mathbf{false}, \mathbf{na}\}$ is called a filtering function.*

Definition 6 (matching function). *Let V be a set of propositional variables. Any function f with domain $D \subseteq \mathcal{I}_V$ and codomain $\{\mathbf{yes}, \mathbf{no}\}$ is called a matching function.*

Any filtering function can be specified with three matching functions in the following way:

Definition 7 (matching functions related to a filtering function). *Let f be a filtering function with domain $D \subseteq \mathcal{I}_V$, $V \in \{v_1, \dots, v_n\}$. The three matching functions related to f are defined as follows:*

- f^{fail} with domain D and codomain $\{\text{yes}, \text{no}\}$, such that for any $I \in D$, $f^{\text{fail}}(I) = \text{yes}$ if and only if $f(I) = \text{fail}$,
- f^{true} with domain $D' = \{I \in D, f(I) \neq \text{fail}\}$, such that for any $I \in D'$, $f^{\text{true}}(I) = \text{yes}$ if and only if $f(I) = \text{true}$,
- f^{false} with domain $D' = \{I \in D, f(I) \neq \text{fail}\}$, such that for any $I \in D'$, $f^{\text{false}}(I) = \text{yes}$ if and only if $f(I) = \text{false}$.

Definition 8 (monotone matching function). *Given the order relation $\text{no} \leq_M \text{yes}$, any matching function f with domain D is said to be monotone if for any $I, J \in D$ such that $I \subseteq J$, $f(I) \leq_M f(J)$.*

Now we will formally define the filtering and the matching functions that are computable using unit resolution.

Definition 9 (propagatable filtering function). *Any filtering function f is said to be propagatable if and only if there exists a propagator which computes f .*

Now, two ways will be considered to compute *matching* functions with unit resolution. The first one consists in using a variable as output under the assumption that \perp is never returned. The second one consists in considering that the output value **yes** when \perp is returned.

Definition 10 (propagatable matching function). *Any matching function f with domain $D \subseteq \mathcal{I}_V$ is said to be propagatable if there exists a propagator $\langle \phi, V, s \rangle$ such that for any $I \in D$, the two following conditions hold:*

1. $\mathcal{U}(\phi|_I) \neq \perp$,
2. $f(I) = \text{yes}$ if and only if $s \in \mathcal{U}(\phi|_I)$.

Definition 11 (ν -propagatable matching function, ν -propagator). *Any matching function f with domain $D \subseteq \mathcal{I}_V$ is said to be ν -propagatable if there exists a CNF formula ϕ such that for any $I \in D$, $f(I) = \text{true}$ if and only if $\mathcal{U}(\phi|_I) = \perp$. The couple $\langle V, \phi \rangle$ is said to be a ν -propagator computing f .*

To end this necessary sequence of definitions, let us address the notion of space complexity of propagatable functions.

Definition 12 (polynomially propagatable functions). *Let \mathcal{F} be a family of filtering functions or a family of matching functions. \mathcal{F} is said to be polynomially propagatable (or polynomially ν -propagatable, if applicable) if and only if any function $f \in \mathcal{F}$ with domain $D \subseteq \mathcal{I}_{\{v_1, \dots, v_n\}}$ can be computed using a CNF formula of size polynomially related to n .*

3.3.2 Propagability versus ν -propagability

In this section, we will show that propagatable and ν -propagatable matching functions have the same expressive power and similar space complexities.

Theorem 3. *Let f be a matching function. f is propagatable if and only if f is ν -propagatable.*

Theorem 4. *Let f be a propagatable matching function. f is polynomially propagatable if and only if f is polynomially ν -propagatable.*

Proof.

1. propagatable $\Rightarrow \nu$ -propagatable.

Let f be a propagatable matching function with domain D , and $P = \langle \phi, V, s \rangle$ be a propagator which computes f . Clearly, for any partial truth assignment $I \in D$, applying unit resolution to the formula $(\phi \wedge (\bar{s}))|_I$ returns \perp if and only if $f(I) = \text{yes}$.

2. ν -propagatable \Rightarrow propagatable.

Let f be a ν -propagatable function with domain $D \subset \mathcal{I}_V$ and ϕ a CNF formula including n variables, such that for any $I \in D$, applying unit resolution to $\phi|_I$ returns \perp if and only if $f(I) = \text{yes}$.

Our aim is to build a new formula ψ such that for any $I \in D$, applying unit resolution to $\psi|_I$ does not return \perp but fixes a variable s to **true** if and only if applying unit resolution to $\phi|_I$ returns \perp , in such a way that the propagator $\langle \psi, V, s \rangle$ computes f .

The formula ψ can be obtained as follows:

$$\psi = \text{reif}(\phi, V) \wedge \left(\bigwedge_{u \in \text{var}(\phi)} (\overline{u_{n+1}^+} \wedge \overline{u_{n+1}^-} \wedge s) \right)$$

The variable s will be fixed to **true** if and only if unit resolution on ϕ fixes both a variable u_i^+ and a variable u_i^- to **true**. According to the theorems 1 and 2, this occurs if and only if applying unit resolution to $\phi|_I$ returns \perp . Then, $\langle \psi, V, s \rangle$ is a propagator which computes f .

Because the two transformations have polynomial space complexity, both theorems 4 and 3 hold. \square

3.3.3 Filtering functions versus matching functions

In this section, we will show that without loss of generality, studying the space complexity of propagatable filtering functions reduces to studying the space complexity of propagatable matching functions.

Theorem 5. *Any filtering function f is propagatable if and only if the three related matching functions f^{true} , f^{false} , and f^{fail} are propagatable.*

Theorem 6. *Any filtering function f is polynomially propagatable if and only if the three related matching functions f^{true} , f^{false} , and f^{fail} are polynomially propagatable.*

Proof.

1. filtering \Rightarrow matching

Let f be a propagatable filtering function and f^{true} , f^{false} , and f^{fail} the related matching functions. Because f is propagatable, there exists a propagator $\langle \phi, V, s \rangle$ that computes f . Then f^{true} , f^{false} , and f^{fail} can be computed with the following propagators, respectively:

- (a) $\langle \phi, V, s \rangle$ (which computes f^{true});
- (b) $\langle \phi \wedge (s \vee t), V, t \rangle$ (which computes f^{false});
- (c) $\langle \psi, V, s^{\text{fail}} \rangle$, where $\langle \psi, V, s^{\text{true}}, s^{\text{false}}, s^{\text{fail}} \rangle$ is the reified counterpart of $\langle \phi, V, s \rangle$.

Clearly, f^{true} , f^{false} , and f^{fail} are propagatable. Now, because the size of ψ is polynomially related to the size of ϕ , if f is polynomially propagatable then f^{true} , f^{false} , and f^{fail} are polynomially propagatable too.

2. matching \Rightarrow filtering

Let f be a filtering function with domain $D \subset \mathcal{I}_V$, $V \in \{v_1, \dots, v_n\}$ and f^{true} , f^{false} , and f^{fail} the related matching functions. Suppose that f^{true} , f^{false} , and f^{fail} are propagatable (polynomially propagatable, respectively). Now let us consider the three following propagators (with formulae of size polynomially related to n , respectively):

- (a) $\langle \phi_1, V, s_1 \rangle$, which computes f^{true} ;
- (b) $\langle \phi_2, V, s_2 \rangle$, which computes f^{false} ;
- (c) $\langle \phi_3, V, s_3 \rangle$, which computes f^{fail} .

Let $\langle \psi_1, V, s_1^{\text{true}}, s_1^{\text{false}}, s_1^{\text{fail}} \rangle$, $\langle \psi_2, V, s_2^{\text{true}}, s_2^{\text{false}}, s_2^{\text{fail}} \rangle$, $\langle \psi_3, V, s_3^{\text{true}}, s_3^{\text{false}}, s_3^{\text{fail}} \rangle$ be the reified counterparts of these propagators. Without loss of generality, let us suppose that, except for the input variables in V , the formulae ψ_1, ψ_2, ψ_3 have no common variable.

The function f can be computed (in polynomial space, respectively) using the following propagator:

$$P = \langle \psi_1 \wedge \psi_2 \wedge \psi_3 \wedge (\overline{s_1^{\text{true}}} \vee s) \wedge (\overline{s_2^{\text{false}}} \vee \overline{s}) \wedge (\overline{s_3^{\text{fail}}}), V, s \rangle$$

Clearly, P computes f , which is then propagatable (polynomially propagatable, respectively). □

3.3.4 Boolean representations

Given $V = \{v_1, \dots, v_n\}$ a set of propositional variables, $D \subseteq \mathcal{I}_V$ a set of partial truth assignments of V , f any matching function with domain D , and I any partial assignment in D , let us define:

- the *Boolean representation* of I as $I_{\mathbb{B}} = (x_1, \dots, x_n, y_1, \dots, y_n) \in \{0, 1\}^n$ such as for any $1 \leq i \leq n$, $x_i = 1$ if and only if $v_i \in I$, and $y_i = 1$ if and only if $\overline{v_i} \in I$,
- the *Boolean representation* of D as $D_{\mathbb{B}} = \{I_{\mathbb{B}}, I \in D\}$,
- the *Boolean representation* of f as $f_{\mathbb{B}} : D_{\mathbb{B}} \mapsto \{0, 1\}$, such that for any $I \in D$, $f_{\mathbb{B}}(I_{\mathbb{B}}) = 1$ if and only if $f(I) = \text{yes}$.

Example 4. The following table gives an example of a matching function f and its Boolean counterpart $f_{\mathbb{B}}$.

I	$I_{\mathbb{B}}$	$f(I)$	$f_{\mathbb{B}}(I_{\mathbb{B}})$
$\{\overline{v_1}, \overline{v_2}\}$	(0, 0, 1, 1)	<i>no</i>	0
$\{\overline{v_1}, v_2\}$	(0, 1, 1, 0)	<i>yes</i>	1
$\{\overline{v_1}\}$	(0, 0, 1, 0)	<i>no</i>	0
$\{v_1, \overline{v_2}\}$	(1, 0, 0, 1)	<i>yes</i>	1
$\{v_1, v_2\}$	(1, 1, 0, 0)	<i>yes</i>	1
$\{v_1\}$	(1, 0, 0, 0)	<i>yes</i>	1
$\{\overline{v_2}\}$	(0, 0, 0, 1)	<i>no</i>	0
$\{v_2\}$	(0, 1, 0, 0)	<i>yes</i>	1
$\{\}$	(0, 0, 0, 0)	<i>no</i>	0

$\langle (\overline{v_1} \vee s) \wedge (\overline{v_2} \vee s), \{v_1, v_2\}, s \rangle$ is a propagator for f .

Example 5. The following table gives a matching function g which is not propagatable, and its Boolean counterpart $g_{\mathbb{B}}$.

I	$I_{\mathbb{B}}$	$g(I)$	$g_{\mathbb{B}}(I_{\mathbb{B}})$
$\{\bar{v}\}$	$(0, 1)$	yes	1
$\{v\}$	$(1, 0)$	no	0
$\{\}$	$(0, 0)$	yes	1

There is no propagator for g because for any formulae $\phi_1 \subseteq \phi_2$, if $\mathcal{U}(\phi_2) \neq \perp$ then any variable fixed by unit resolution on ϕ_1 will be fixed on ϕ_2 as well. It follows that the third line of the table is not consistent with the second one.

3.4 Synthesis

In this section, we first introduced the notion of filtering function as a general model of functions that can be computed by unit resolution. We then showed that any filtering function reduces to three matching functions, which are functions with binary codomain ($\{\text{yes}, \text{no}\}$ without loss of generality) that can either be computed by unit resolution in two different ways: (1) unit resolution detects an inconsistency when the output value is **yes**, (2) it fixes a predefined output variable to **true** when the output value is **yes**. The main result of this section is that without loss of generality, studying the expressive power of unit resolution can be reduced to studying the tractability and the complexity of computing *matching functions* with unit resolution. As a corollary, in the sequel of the paper, only propagatable matching functions will be considered.

4 Expressive power of propagators

Using a *complete* truth assignment as input values, unit resolution has the same expressive power as Boolean circuits, because elementary gates can be directly translated into clauses. In this section, we will show that if some input variables are not fixed, the expression power of unit resolution fall down to the expression power of *monotone* Boolean circuits, i.e., circuits with only **or** / **and** gates.

4.1 Boolean circuits

A Boolean circuit is a directed acyclic graph representing a Boolean formula. It is said to be *monotone* when it contains only **and** and **or** gates.

In the following, a Boolean circuit will be represented by a triplet $\langle L, G, w \rangle$, where L is a set of input labels, w is the output label, and G is a set of gates. A **or** gate (**and** gate, respectively) is denoted $\text{or}(E, t)$ (**and**(E, t), respectively), where E is the set of input labels and t is the output label of the gate. A **not** gate is denoted $\text{not}(q, t)$, where q is the input label of the gate and t is its output label.

Given a Boolean circuit $C = \langle \{e_1, \dots, e_n\}, G, w \rangle$ and any $x = (x_1, \dots, x_n) \in \{0, 1\}^n$, let $C(x)$ denote the output value of C under the assumption that its input values are x_1, \dots, x_n . Formally, $C(x)$ can be defined as $\text{val}(w)$ such that for any $1 \leq i \leq n$, $\text{val}(e_i) = x_i$, for any gate $\text{or}(E, t) \in G$, $\text{val}(t) = \bigvee_{e \in E} \text{val}(e)$, for any gate **and**(E, t) $\in G$, $\text{val}(t) = \bigwedge_{e \in E} \text{val}(e)$, and for any gate $\text{not}(q, t) \in G$, $\text{val}(t) = \neg \text{val}(q)$.

For convenience, an additional gate **tie**(q, t) will be used to connect two nodes q and t in such a way that $\text{val}(t) = \text{val}(q)$.

Given any Boolean function f with domain $D \subseteq \{0, 1\}^n$ and codomain $\{0, 1\}$, any Boolean circuit C with n inputs is said to *compute* f if and only if for any $x \in D$, $C(x) = f(x)$.

4.2 Circuits computing propagatable functions

Because the Boolean counterpart of any matching function is a Boolean function, it can be computed by a Boolean circuit. In this section, we will show that any matching function is propagatable if and only if its Boolean counterpart can be computed using a *monotone* circuit. Furthermore, we will establish a relationship between space complexity of propagatable matching functions and monotone circuit complexity.

Theorem 7. *For any matching function f , if there exists a monotone circuit with n gates, each of them with at most k inputs, which computes $f_{\mathbb{B}}$, then there exists a propagator with $O(nk)$ clauses, which computes f .*

Proof. Let us consider any matching function f with domain $D \subseteq \mathcal{I}_V$, $V = \{v_1, \dots, v_n\}$, and any monotone circuit $Q = \langle L, G, u_k \rangle$ computing $f_{\mathbb{B}}$.

Without loss of generality, let us suppose that

- the set of input labels of Q is $L = \{e_1, \dots, e_{2n}\}$,
- the set of the output labels of the gates of Q is $\{u_1, \dots, u_k\}$.

Let τ be a function that maps the labels of Q to propositional literals such that

$$\begin{cases} \tau(e_i) = v_i, 1 \leq i \leq n \\ \tau(e_i) = \overline{v_{i-n}}, n+1 \leq i \leq 2n \\ \tau(u_i) = v_{n+i}, 1 \leq i \leq k-1 \\ \tau(u_k) = s \end{cases}$$

For any gate $g = \mathbf{and}(\{\alpha_1, \dots, \alpha_m\}, t) \in G$, let $\pi(g) = (\overline{\tau(\alpha_1)} \vee \dots \vee \overline{\tau(\alpha_m)} \vee \tau(t))$.

For any gate $g = \mathbf{or}(\{\alpha_1, \dots, \alpha_m\}, t) \in G$, let $\pi(g) = \bigwedge_{i=1}^m (\tau(\alpha_i) \vee \tau(t))$.

Let

$$\phi = \bigwedge_{g \in G} \pi(g).$$

Now let us show by induction on the number k of gates in Q that the propagator $P = \langle \phi, V, s \rangle$ computes f .

The property holds for $k = 0$ because if the circuit Q has no gate, the output label is one of the input labels e_i or e_{i+n} related to the input variable $v_i \in V$. If the input label is e_i then the propagator $\langle \{\}, V, v_i \rangle$ computes f . If the input label is e_{i+n} then the propagator $\langle (v_i \vee s), V, s \rangle$, where s is a new fresh variable, computes f .

Now, let us suppose the the property holds for any circuit with less than k clauses, $k > 0$. Let $Q = \langle L, G, u \rangle$ be any k -gates monotone Boolean circuit which computes the Boolean counterpart $f_{\mathbb{B}}$ of f with input variables $\{v_1, \dots, v_n\}$. Let g be the output gate of Q . Let $\alpha_1, \dots, \alpha_m$ be the input labels of g . For any $1 \leq i \leq m$, let $Q_i = \langle L, G \setminus \{g\}, \alpha_i \rangle$. By induction hypothesis, each Q_i computes the Boolean counterpart $f_{\mathbb{B}_i}$ of the matching function f_i computed by the propagator $P_i = \langle \phi \setminus \pi(g), V, \tau(\alpha_i) \rangle$.

Let us consider two cases:

1. The output gate of Q is $g = \mathbf{and}(\{\alpha_1, \dots, \alpha_m\}, u)$.

Because of the nature of g , for any $I \in \mathcal{I}_V$, $f_{\mathbb{B}}(I) = 1$ if and only if for any $1 \leq i \leq m$, $f_i(I_{\mathbb{B}}) = 1$. Because of the nature of $\pi(g)$, $f(I) = \mathbf{yes}$ if and only if for any $1 \leq i \leq m$, $\tau(\alpha_i) \in \mathcal{U}((\phi \setminus \pi(g))|_I)$. Then $f(I) = \mathbf{yes}$ if and only if $f_{\mathbb{B}}(I_{\mathbb{B}}) = 1$.

2. The output gate of Q is $g = \mathbf{or}(\{\alpha_1, \dots, \alpha_m\}, u)$.

Because of the nature of g , for any $I \in \mathcal{I}_V$, $f_{\mathbb{B}}(I) = 1$ if and only there exists $1 \leq i \leq m$, such as $f_i(I_{\mathbb{B}}) = 1$. Because of the nature of $\pi(g)$, $f(I) = \mathbf{yes}$ if and only if there exists $1 \leq i \leq m$, such that $\tau(\alpha_i) \in \mathcal{U}((\phi \setminus \pi(g))|_I)$. Then $f(I) = \mathbf{yes}$ if and only if $f_{\mathbb{B}}(I_{\mathbb{B}}) = 1$.

□

Example 6. The circuit of the figure 1 can be translated into a CNF formula ϕ in the following way:

- the gate $\text{and}(\{e_1, e_2\}, u_1)$ produces the clause $(\bar{v}_1 \vee \bar{v}_2 \vee v_3)$;
- the gate $\text{or}(\{u_1, e_4\}, u_2)$ produces the clauses $(\bar{v}_3 \vee s)$ and $(v_2 \vee s)$.

This circuit computes the Boolean counterpart $f_{\mathbb{B}}$ of the function f computed by the propagator $\langle \phi, \{v_1, v_2\}, s \rangle$.

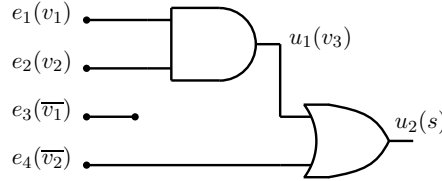


Figure 1: A monotone circuit computing the Boolean counterpart of a propagatable function.

Theorem 8. For any matching function f , if there exists a propagator $\langle \phi, V, s \rangle$ computing f , then there exists a monotone circuit with $O(n^2k)$ gates computing $f_{\mathbb{B}}$, where n is the number of variables and k the number of clauses in ϕ .

Proof. Let $\langle \phi, V, s \rangle$ be a propagator computing a matching function f . Clearly, the propagator $P = \langle \psi = \text{reif}(\phi, V), V, s_{n+1}^+ \rangle$ computes f too. According to Lemma 1 and Theorem 1, ψ can be decomposed as $\psi_0 \wedge \psi_1 \wedge \dots \wedge \psi_{n+1}$ such that

- ψ_0 contains the initialization clauses of rank 0 of the reified counterpart of ϕ ,
- ψ_1 contains the initialization clauses of rank 1 as well as the injection clauses,
- for any $2 \leq i \leq n$, ψ_i contains both the propagation clauses and the deduction clauses of rank i .

The corresponding circuit Q will contain the following nodes:

- two *input nodes* $\diamond t$ and $\diamond \bar{t}$ related to each input variable $t \in V$, with the convention that $\diamond t = 1$ if and only if t is assigned to **true**, and $\diamond \bar{t} = 1$ if and only if t is assigned to **false**;
- one *major node* $\diamond v$ related to any variable $v \in \text{var}(\psi)$ that can be assigned to **true** by unit resolution, with the convention that $\diamond v = 1$ if and only if unit resolution fixes v to **true**;
- some *additional nodes*, if applicable;

Major nodes and additional nodes can be constant, i.e. permanently assigned either to 0 or 1. The constant nodes are not explicitly represented in the circuit but are referenced in the sets U_0 and U_1 , respectively.

The circuit Q consists of several layers Q_1, \dots, Q_{n+1} , where each Q_i simulates the stage i of unit resolution on $\psi|_I$ for any $I \in \mathcal{I}_V$.

At the first step of the construction, U_0 is initialized with the nodes $\diamond v_0^+$ ($\diamond v_0^-$, respectively) for any variable $v \in \text{var}(\phi)$ such that (v_0^+) ((v_0^-) , respectively) does not occur in ψ_0 , and U_1 is initialized with the nodes $\diamond v_0^+$ ($\diamond v_0^-$, respectively) for any variable v_0^+ (v_0^- , respectively) such that the clause (v_0^+) ((v_0^-) , respectively) occurs in ψ_0 .

Each of the next steps builds Q_i in such a way that it simulates the effect of unit resolution applied to ψ_i . This is done as follows:

For each variable v of ϕ and for each variable $u \in \{v_i^+, v_i^-\}$, let C be the set of clauses of ψ_i containing u , simplified by removing the clauses containing a literal \bar{w} such that $\diamond w \in U_0$ and removing any literal \bar{w} such that $\diamond w \in U_1$ from the other clauses.

If the set C is empty, which means that unit resolution cannot fix u , then $\diamond u$ is added to U_0 . If C contains a clause (u) , which means that unit resolution will always fix u to **true**, then $\diamond u$ is added to U_1 . If C contains only one clause $(\bar{w} \vee u)$, meaning that u is fixed to **true** if and only if w is previously fixed to **true**, the connection $\text{tie}(\diamond w, \diamond u)$ is produced. If C contains only one clause with more than two literals like $(\bar{w}_1 \vee \dots \vee \bar{w}_k \vee u)$, meaning that u is fixed to **true** if and only if w_1 and ... and w_k are previously fixed to **true**, the gate $\text{and}(\{\diamond w_1, \dots, \diamond w_k\}, \diamond u)$ is produced.

In the other cases, i.e., when there are several clauses which can allow unit resolution to fix u , an additional node α_c is created for each clause $c \in C$. For any binary clause $(\bar{w} \vee u) \in C$, the gate $\text{tie}(\diamond w, \alpha_c)$ is produced. For any other clause $(\bar{w}_1 \vee \dots \vee \bar{w}_k \vee u)$, the gate $\text{and}(\{\diamond w_1, \dots, \diamond w_k\}, \alpha_c)$ is produced. Then the gate $\text{or}(\{\alpha_c, c \in C\}, \diamond u)$ is produced, in such a way that $\text{val}(\diamond u) = 1$ if and only if unit resolution fixes u to **true**.

Because each sub-circuit Q_i simulates exactly the effect of unit resolution on the corresponding formula ψ_i , the value of the output node $\diamond s_{n+1}^+$ will reflect the value of the output variable s_{n+1}^+ after all propagation stages on ψ have been made.

The number of gates in the circuit is linearly related to the number of clauses in the reified counterpart of ϕ , which is $O(n^2k)$. \square

Example 7. Let us consider the propagator $\langle (a \vee \bar{b} \vee c), \{a, b\}, c \rangle$. At the first stage of the construction, $U_0 = \{\diamond c_0^+, \diamond c_0^-\}$, and $U_1 = \{\}$ because ψ_0 is empty. The input nodes of the circuit are $\diamond a, \diamond \bar{a}, \diamond b$, and $\diamond \bar{b}$.

The first layer of the circuit is based on:

$$\psi_1 = \overbrace{(\bar{a} \vee a_1^+) \wedge (a \vee a_1^-) \wedge (\bar{b} \vee b_1^+) \wedge (b \vee b_1^-)}^{\text{injection clauses}}$$

It consists in the connections $\text{tie}(\diamond a, \diamond a_1^+)$, $\text{tie}(\diamond \bar{a}, \diamond a_1^-)$, $\text{tie}(\diamond b, \diamond b_1^+)$, $\text{tie}(\diamond \bar{b}, \diamond b_1^-)$. The variables $\diamond c_1^+$ and $\diamond c_1^-$ are added to U_0 .

The second layer is based on:

$$\psi_2 = \overbrace{(\bar{a}_1^+ \vee a_2^+) \wedge (\bar{a}_1^- \vee a_2^-) \wedge (\bar{b}_1^+ \vee b_2^+) \wedge (\bar{b}_1^- \vee b_2^-) \wedge (\bar{c}_1^+ \vee c_2^+) \wedge (\bar{c}_1^- \vee c_2^-)}^{\text{propagation clauses}} \wedge \underbrace{(\bar{a}_1^- \vee \bar{b}_1^+ \vee c_2^+) \wedge (\bar{a}_1^- \vee \bar{c}_1^- \vee b_2^-) \wedge (b_1^+ \vee \bar{c}_1^- \vee a_2^+)}_{\text{deduction clauses}}$$

The two last propagation clauses and the two last deduction clauses are ignored because $\diamond c_1^+$ and $\diamond c_1^-$ are in U_0 . The four first propagation clauses are translated into $\text{tie}(\diamond a_1^+, \diamond a_2^+)$, $\text{tie}(\diamond a_1^-, \diamond a_2^-)$, $\text{tie}(\diamond b_1^+, \diamond b_2^+)$, $\text{tie}(\diamond b_1^-, \diamond b_2^-)$. The first deduction clause is translated into the gate $\text{and}(\{a_1^-, b_1^+\}, c_2^+)$. c_2^- is added to U_0 .

The third layer is based on:

$$\psi_3 = \overbrace{(\bar{a}_2^+ \vee a_3^+) \wedge (\bar{a}_2^- \vee a_3^-) \wedge (\bar{b}_2^+ \vee b_3^+) \wedge (\bar{b}_2^- \vee b_3^-) \wedge (\bar{c}_2^+ \vee c_3^+) \wedge (\bar{c}_2^- \vee c_3^-)}^{\text{propagation clauses}} \wedge \underbrace{(\bar{a}_2^- \vee \bar{b}_2^+ \vee c_3^+) \wedge (\bar{a}_2^- \vee \bar{c}_2^- \vee b_3^-) \wedge (b_2^+ \vee \bar{c}_2^- \vee a_3^+)}_{\text{deduction clauses}}$$

The propagation clause $(\overline{c_2^+} \vee c_3^+)$ is translated into the connection $\mathbf{tie}(\diamond c_2^+, \alpha_1)$, the deduction clause $(\overline{a_2^-} \vee \overline{b_2^+} \vee c_3^+)$ is translated into the gate $\mathbf{and}(\{a_2^-, b_2^+\}, \alpha_2)$, and the clause $\mathbf{or}(\{\alpha_1, \alpha_2\}, \diamond c_3^+)$ is added, in such a way that $\diamond c_3^+$ is set to 1 either if $\diamond c_2^+$ is set to 1 or if both $\diamond a_2^-$ and $\diamond b_2^+$ are set to 1, i.e., if at stage 2 of unit resolution, either c is fixed to **true** or a and b are fixed to **false** and **true**, respectively...

A part of the corresponding circuit is given Figure 2. (Recall that this circuit is obtained from a reified formula, which, as mentioned above, presents some redundancies.)

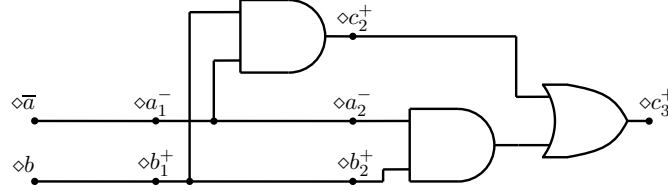


Figure 2: A part of the circuit related to the example 7.

Theorem 9. Let f be any matching function with domain D . f is propagatable if and only if it is monotone.

Proof. Recall that any Boolean function h with domain D_h is said to be monotone if for any $z, t \in D_h$, if $z \leq_B t$ then $h(z) \leq_B h(t)$, where the ordering relation \leq_B is defined as follows: $0 \leq_B 1, 0 \leq_B 0, 1 \leq_B 1$, $(z_1, \dots, z_n) \leq_B (t_1, \dots, t_n)$ if and only if $z_i \leq_B t_i, 1 \leq i \leq n$.

Now, given any matching function f with domain D , because for any $I, J \in D$, $I \subseteq J$ if and only if $I_{\mathbb{B}} \leq_B J_{\mathbb{B}}$, $f_{\mathbb{B}}$ is monotone on $D_{\mathbb{B}}$ if and only if f is monotone on D .

Let f be any monotone matching function with domain D . Because $f_{\mathbb{B}}$ is monotone, it can be computed by a monotone circuit. It follows from Theorem 7 that f is propagatable.

Now let us consider any propagatable matching function f with domain D . It follows from Theorem 8 that $f_{\mathbb{B}}$ can be computed by a monotone circuit, which implies that $f_{\mathbb{B}}$ is monotone. Then f is monotone. \square

Theorem 10. Any family \mathcal{F} of propagatable functions is propagatable in polynomial space if and only if the family of the Boolean counterparts of \mathcal{F} has polynomial space monotone circuit complexity, i.e., these functions can be calculated by monotone circuits with a polynomial number of gates.

Proof. The proof of theorem 7 shows how to create a propagator from a monotone circuit. Each **and** gate with n inputs is translated into one n -ary clause, and each **or** gate with n inputs is translated into n binary clauses.

The proof of theorem 8 shows how to create a monotone circuit from a propagator $P = \langle \phi, V, s \rangle$. This circuit is based on the reified counterpart ψ of the formula ϕ . Each clause of ψ with n literals is involved in at most n **and** gates, and each literal is involved in at most one **or** gate. \square

5 Synthesis and perspectives

Altogether, the results given in this paper provide important information about the expressive power of unit resolution. In particular, we can show that there exist polynomial time complexity propagatable functions that admit only propagators with an exponential number of clauses.

As an example, let us consider the Boolean functions $\text{pm}^{(n)}$, like *perfect matching*, such that for any n -bits Boolean encoding g of a graph G , $\text{pm}^{(n)}(g) = 1$ if and only if there exists a perfect matching for G , that is a set of edges that covers each vertex exactly once.

Next, let us consider the variants $\text{vpm}^{(n)}$ such that

- the domain $D_{\text{vpm}^{(n)}}$ of $\text{vpm}^{(n)}$ is the set $\{(x_1, \dots, x_n, 0, \dots, 0), (x_1, \dots, x_n) \in D_{\text{pm}^{(n)}}\}$, where $D_{\text{pm}^{(n)}}$ is the domain of $\text{pm}^{(n)}$,
- for any $b = (x_1, \dots, x_n, 0, \dots, 0) \in D_{\text{vpm}^{(n)}}$, $\text{vpm}^{(n)}(b) = 1$ if and only if $\text{pm}^{(n)}(x_1, \dots, x_n) = 1$.

Now, let $\text{fpm}^{(n)}$ denote the matching functions related to $\text{vpm}^{(n)}$. It is known that $\text{pm}^{(n)}$, then $\text{vpm}^{(n)}$, have polynomial time computational complexity but exponential monotone circuit complexity [15]. It follows from theorem 8 that $\text{fpm}^{(n)}$ are filtering functions requiring an exponential number of clauses to be computed using unit resolution.

This means that although unit resolution has the same expression power as Boolean circuits regarding Boolean functions, it has a lower expression power, namely the expression power of monotone circuits, regarding filtering functions.

This is both very interesting and annoying, because in SAT solvers unit propagation operates on filtering functions rather than Boolean functions. Maybe this potential weakness of unit resolution can be compensated for by other speed-up technologies. As a research perspective, this has to be verified. Meanwhile, in the field of encoding constraints into CNF, it would be very relevant to determine which problems can be solved as efficiently using a *simple* SAT solver, under CNF encoding, as using a dedicated constraint solver maintaining generalized arc consistency. This supposes knowing the complexity of the related filtering functions regarding unit resolution.

At least two research directions follow from the ideas presented in this paper. The first one is to characterize the expression power of some other speed-up techniques used in modern SAT solvers, like clause learning. The second one consists in the research of deduction techniques that can polynomially compute any CNF encoded polynomial time complexity filtering functions.

References

- [1] Fahiem Bacchus. GAC via unit propagation. In Christian Bessiere, editor, *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 2007.
- [2] Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP-03)*, pages 108–122, 2003.
- [3] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. A translation of pseudo boolean constraints to sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:191–200, 2006.
- [4] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New encodings of pseudo-boolean constraints into cnf. In *Theory and Applications of Satisfiability Testing - SAT 2009 (SAT'09)*, pages 181–194, 2009.
- [5] Niklas Eén and Niklas Soörensön. An extensible sat-solver. In *Proceedings of SAT 2003*, pages 202–518, 2003.
- [6] Ian P. Gent. Arc consistency in sat. In *Proceedings of ECAI 2002*, 2002.
- [7] E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat solver. In *Proc. of DATE 2002*, pages 142–149, 2002.
- [8] Leslie M. Goldschlager. The monotone and planar circuit value problems are log space complete for p. *SIGACT News*, 9(2):25–29, 1977.

- [9] L. Henschen and L. Wos. Unit refutations and horn sets. *J. ACM*, 21(4):590–605, 1974.
- [10] Neil D. Jones and William T. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3(1):105 – 117, 1976.
- [11] Chu Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In Gert Smolka, editor, *Principles and Practice of Constraint Programming-CP97*, volume 1330 of *Lecture Notes in Computer Science*, pages 341–355. Springer Berlin / Heidelberg, 1997. 10.1007/BFb0017450.
- [12] Joao Marques-Silva and Inês Lynce. Towards robust cnf encodings of cardinality constraints. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP-07)*, pages 483–497, 2007.
- [13] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *39th Design Automation Conference*, June 2001.
- [14] Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP-05)*, pages 827–831, 2005.
- [15] E. Tardos. The gap between the monotone and non monotone circuit complexity is exponential. *Combinatorica*, 8:141–142, 1988.
- [16] J. P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.